TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Implementació d'un ambient Network amb P2P-DHT

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: David González Jiménez

DIRECTOR: Roc Messeguer Pallarés

DATA: 7 de març de 2008

Títol: Implementació d'un ambient Network amb P2P-DHT

Autor: David González Jiménez

Director: Roc Messeguer Pallarés

Data: 7 de març de 2008

Resum

El propósito principal de este proyecto, es desarrollar una aplicación distribuida que simule un Ambient Network, en el que los diferentes elementos realicen procesos de autoconfiguración y autogestión únicamente con su integración a la red.

La aplicación deberá, por tanto, mantener actualizado el número de elementos que se encuentra en la red, establecer comunicaciones entre los nodos para mantener la información actualizada y tomar decisiones a través de los resultados ofrecidos por otros elementos en la red.

Se ha llevado a cabo un estudio previo acerca de las tecnologías Freepastry y Past, para estudiar la viabilidad de uso en este proyecto. En este estudio, se han realizado pruebas a partir de los tutoriales expuestos por los desarrolladores de ambas propuestas.

Una vez finalizado el estudio, ambas tecnologías se han considerado aptas para el desarrollo del proyecto. Estas tecnologías crean un entorno de red virtual distribuido donde cada uno de los participantes puede interactuar directamente, independientemente de su situación.

Tras el estudio en profundidad de las dos tecnologías, se ha realizado el diseño de la aplicación deseada, para posteriormente proceder al desarrollo de la misma mediante el lenguaje orientado a objetos Java. Este lenguaje facilita una integración sencilla y rápida de diferentes tecnologías.

Finalmente y tras una batería de pruebas para comprobar el correcto funcionamiento, se ha obtenido una aplicación que permite, tal como se comentaba inicialmente, a través de una tecnología P2P, simular un ambiente domótico en el que integrar procesos de configuración y gestión en cada uno de los componentes.

Title: Implementation of a P2P-DHT Ambient Network

Author: David González Jiménez

Director: Roc Messeguer Pallarés

Date: March, 8th 2008

Overview

The main purpose of this project is to develop a distributed application to simulate Ambient Network, in which various elements execute self-configuration and self-management processes only with a joint in the network.

The application will have to keep, therefore, updated the number of elements founded in the network, to establish communications between the nodes to keep updated the information and take decisions across the results offered by other elements in the network.

A previous research in the Freepastry and Past technologies has been required, both of them used in the development of this project. In this study, the tutorials suggested by the two developer teams have been tested previously. The exposed technologies establish a distributed virtual network where each participant can interact directly, with independence in its situation.

Afterwards, the design of the application required has been made, Then, the implementation has been performed by the object oriented language, Java. This language provides a simply and quick integration among different technologies.

Once finished the study, both technologies have been considered to be suitable for the development of the project. These technologies create an virtual network environment distributed where each of the participants can interact directly, independently of his situation.

Finally, a battery of tests has been done to prove a well operation. The application made allows the simulation, by mean of P2P technology, of a Ambient Network in which the configuration and management processes are integrated in each component.

A mi familia por darme su respaldo en estos años.

A todos los amigos y compañeros por amenizar el paso por esta universidad.

Al Director del proyecto Roc Messeguer, porque su guía y su paciencia han hecho posible este proyecto.

A Raquel porque su alegría y desenfado han iluminado mi camino.

A todos ellos, gracias.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

En este primer capítulo se explican las razones y motivaciones para la realización de este proyecto. También se hace hincapié en los objetivos que se fijaron para la realización de este trabajo.

El capítulo segundo recoge el escenario de trabajo. En él se definen las diferentes tecnologías utilizadas en el proyecto y se realiza una explicación de su funcionamiento y propiedades.

En el tercer capítulo se encuentran todos los datos referentes al diseño de la aplicación. Se definen, por tanto, los diferentes elementos en la red y todas las características que los forman. Comenzando por los procesos de inicio hasta el final de su ejecución, incluyendo diagramas de clases donde poder observar el diseño de la aplicación con más detalle y poder observar la interacción entre las diferentes clases programadas, son temas que se tratan en este capítulo.

El capítulo cuarto engloba una serie de pruebas realizadas con la aplicación. En él se explican las diferentes pruebas en diferentes escenarios y los resultados obtenidos.

Las conclusiones recogidas durante la elaboración de este proyecto se recogen en el capítulo 5, donde se evalúan. También se exponen diferentes puntos de vista sobre las dificultades experimentadas en la realización del proyecto.

1.1. Razón y oportunidad del proyecto

La rápida evolución de las tecnologías de la comunicación y la amplia red de servicios necesaria para el mantenimiento de ésta, originaron las tecnologías descentralizadas. Esto permite que la sobrecarga de los sistemas no sea un problema y que el número de usuarios con acceso a cualquier tipo de recurso sea ampliable.

Hoy en día, las tecnologías Peer-to-Peer se encuentran muy extendidas en todos los ámbitos, y cada vez más, se intentan integrar estas características como solución de acceso a recursos publicados.

Un claro ejemplo de la integración de este tipo de tecnologías en el mundo de las comunicaciones, es el uso que se les está dando en el ámbito de la compartición de ficheros o en los sistemas de mensajería. En ambos casos, el número de usuarios que utilizan estos servicios es un número elevado que sigue creciendo.

En este proyecto, se define un único usuario. Se pretende establecer un sistema de autoconfiguración y autogestión en el que los equipos actúen tomando decisiones a través de la información que intercambian, y que este único usuario, pueda modificar las decisiones tomadas por los equipos.

1.2. Objetivos del proyecto

El objetivo de este proyecto consiste en estudiar el comportamiento de la tecnología Freepastry. Para ello, se ha implementado una aplicación, que simula un ambiente domótico, que permite la autogestión i configuración de los elementos necesarios para la manipulación de la iluminación de un habitáculo.

Como primer objetivo a alcanzar, se establece la familiarización con los entornos P2P. Para ello, se realiza un estudio exhaustivo de Freepastry mediante el cual se pueden apreciar tanto las propiedades de los sistemas P2P, como los conceptos, las características y las principales opciones que ofrece Freepastry. Este proceso permite adquirir los conocimientos necesarios para establecer un escenario basado en esta tecnología y poder trabajar con ella.

El segundo objetivo, es la prueba de algunas aplicaciones y códigos de ejemplo, para observar el comportamiento y comprobar las funcionalidades apreciadas en el estudio previo. Estas pruebas permitirán obtener conocimientos más específicos, tales como, el uso de los recursos y servicios proporcionados por la tecnología y el funcionamiento de los protocolos.

A partir de aquí, y como último objetivo se diseña la aplicación utilizando la técnica de diseño UML, implementándola a partir de los ejemplos vistos anteriormente.

CAPÍTULO 2. ESCENARIO DE TRABAJO

A lo largo de este capítulo se definen las tecnologías y aplicativos utilizados en el escenario del proyecto. El entorno de red sobre el que se diseña es un modelo Peer-to-Peer, denominado FreePastry.

2.1. Peer-to-Peer

2.1.1. Especificaciones de la tecnología Peer-to-Peer

Una red Peer-to-Peer, conocida más comúnmente como P2P, define una arquitectura de red descentralizada y distribuida. Este conjunto de características se obtiene de la desaparición de clientes y servidores como grupos independientes, ya que, P2P basa su arquitectura en un esquema de nodos que se comportan simultáneamente como cliente y servidor.

La tecnología P2P se antepone a la arquitectura cliente/servidor, donde el solicitante y el proveedor de datos tienen unas funcionalidades fijadas por definición. P2P, en cambio, no ofrece un esquema de servidor central que controla o sirve a los demás, sino que todos los nodos interactúan como cliente y servidor con el resto de entidades de la red, teniendo todas ellas el mismo protagonismo, como podemos observar en la figura 2.1.

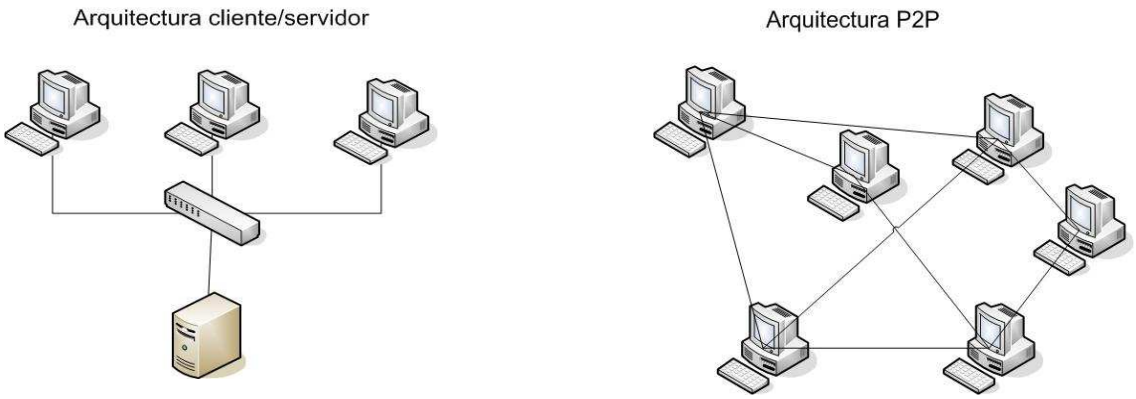

Fig. 2.1. Modelos de arquitectura

Estas características permiten que, aunque un nodo abandone la red, bien por una desconexión involuntaria o bien por un abandono de la red, otro nodo de la red haga las funciones requeridas.

Desde un punto de vista intuitivo, cada nodo es capaz de descubrir otros nodos en la red y conectarse con ellos sin que exista una relación previa entre ellos.

2.1.2. Características de P2P

Las principales características de los sistemas P2P son evitar la aparición de cuellos de botella en los nodos, la resistencia a fallos y la flexibilidad a la adaptación de entradas y salidas de nodos en el grupo.

Estas características se producen gracias a la dispersión de nodos, típica de esta arquitectura, que evita la formación de saturación (efecto conocido como cuello de botella) para acceder a servicios. Lo mismo ocurre con la flexibilidad y la incorporación/abandono del grupo proporcionada por P2P. Si se tuviera una arquitectura modelo cliente/servidor, el efecto de una caída puede tener consecuencias fatales en el caso de un servidor, mientras que un acceso o abandono de una red de este tipo conlleva que previamente se haya realizado una configuración cuidadosa del equipo.

2.2. Distributed Hash Tables

Para analizar las Tablas de Hash Distribuido (Distributed Hash Tables, DHT), es necesario en primer lugar mencionar que es una tabla de Hash.

Se utilizan en aplicaciones de identificación de ficheros o puestos de trabajo (como podría ser el P2P), en comprobaciones de ficheros y también en bases de datos. Esta tecnología es usada por diferentes protocolos de uso extendido como *CAN*, *Chord* o *Kademlia* implementados en aplicaciones del tipo de *BitTorrent*, *The Circle*, *Emule*, *JXTA*, *I2P* u *Overnet*.

La principal explotación de este tipo de infraestructuras viene dada por sistemas de ficheros distribuidos, sistemas de compartición de archivos, almacenamiento cooperativo en Web, multidifusión y servicios DNS.

A grandes rasgos, una tabla de Hash es un método por el cual se asigna un identificador a un objeto. El identificador se genera aplicando una serie de complejas operaciones matemáticas sobre diferentes rasgos del objeto, como podrían ser el recurso compartido, el nombre, el tamaño o incluso la fecha de creación. Entre los algoritmos más habituales para la creación de identificadores se encuentra MD5. Md5 es un código de 128 bits que se representa típicamente como un número de 32 dígitos hexadecimal

En una red DHT, tal como se muestra en la figura 2.2, se ofrece un sistema distribuido y descentralizado que reparte una serie de claves entre los nodos, permitiendo una eficiente transmisión de información entre el cliente y el propietario de la clave.

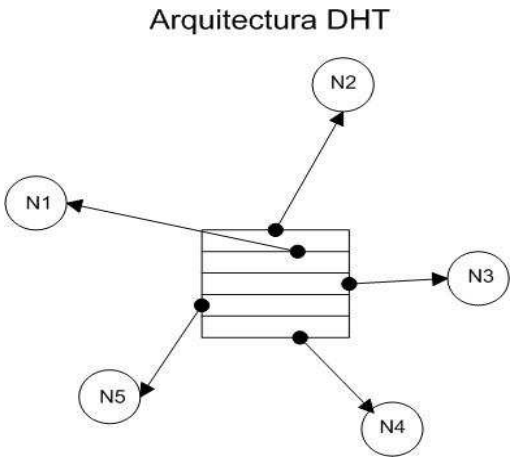

Fig. 2.2. Arquitectura DHT

Este tipo de redes se diseña para tratar un gran número de nodos y gestionar sus entradas y salidas del grupo.

2.3. FreePastry

La aplicación en Java FreePastry permite implementar aplicaciones P2P basadas en el esquema de red DHT, anteriormente comentado. La topología presentada por Freepastry es en anillo, rompiendo totalmente con las topologías habituales de P2P.

2.3.1. Estructura de la red FreePastry

FreePastry ofrece una topología lógica en anillo, la cual trabaja sobre redes como intranet, LAN o internet. Esto sitúa a FreePastry por encima de la capa de Transporte de la pila protocolaria TCP/IP.

La entrada de un nuevo nodo en el anillo lógico de FreePastry provoca una reestructuración de los nodos. Si se produce una entrada al grupo, FreePastry otorga una posición aleatoria en el anillo y reestructura al resto de los nodos, informando a los nodos colindantes de la aparición de un nuevo vecino. Esta función se realizada por el NodeHandle explicada a continuación con más detalle.

En caso de abandono, se informa a los nodos vecinos de que ha abandonado un vecino suyo para que restablezcan el camino y mantengan conectividad entre ellos, asegurando la conectividad del sistema.

2.3.2. Terminología y entidades en FreePastry

Si nos adentramos en la estructura de la red, encontramos los siguientes términos y funcionalidades de los nodos, necesarias para complementar el entendimiento de este tipo de red.

- **NodeId:** Identificador aleatorio asignado a cada uno de los nodos en la arquitectura Freepastry.
- **NodeIdFactory:** Es el proceso encargado de generar un NodeId local. En una implementación real de Pastry, se considera crítico no poder escoger el propio NodeId. Para cumplir esto, es posible requerir un certificado para centralizar la asignación de los identificadores de nodo.
- **PastryNode:** Es un nodo en la red. La aplicación enviará mensajes a través del nodo y el nodo entregará los mensajes a la aplicación.
- **NodeHandle:** Es una referencia específica al PastryNode en la red. El NodoHandle contiene el NodeId y la información de la capa de transporte (dirección IP y puerto) necesaria para encontrar el nodo. El NodeHandle se usa para agregar el nodo a la red.
- **BootstrapNode:** Es el componente usado para acceder a la red. Cuando un nodo se inicializa, puede tanto acceder a un anillo existente como crear uno nuevo. Una vez el primer nodo está operativo, lo lógico es que todos los nuevos nodos accedan al anillo ya creado.
- **PastryNodeFactory:** Construye e inicializa PastryNode, incluyendo la capa de transporte y la tabla de rutas. Se hace necesario para mantener adecuadamente la estructura de la red.

2.3.3. Método de comunicación entre nodos

Pastry es un algoritmo de búsqueda y enrutado basado en DHT, por ello, cada nodo de la red FreePastry tiene un identificador de 128 bits. Este identificador está generado mediante una tabla de Hash.

Los nodos configuran sus identificadores en función de los nodos cercanos, siendo así las claves de dos nodos cercanos o con acceso a los mismos datos, muy similares en los primeros dígitos de su identificador. El identificador, básicamente nos informa de que datos tenemos acceso desde él.

Tal como se ha comentado anteriormente, FreePastry construye un anillo lógico sobre una red con otro tipo de topología. Al no existir conectividad directa entre los nodos del sistema, la comunicación entre nodos adyacentes

es bastante sencilla, pero cuando aparece más de un salto de nodo se presenta un esquema más complejo.

El algoritmo de Freepastry redirige el mensaje a enviar al nodo que contiene el identificador más similar numéricamente a la clave. Suponiendo que la red, conste de N nodos, este algoritmo puede enrutar a cualquier nodo en $log_{2^b} N$ pasos, siendo b un valor fijado teóricamente.

Las tablas de enrutado de cada nodo, contienen únicamente, $(2^b - 1) \cdot (log_{2^b} N)$ entradas, donde en cada entrada se asocia el identificador de nodo con la IP de éste. La tabla de rutas está organizada con $log_{2^b} N$ filas y con $2^b - 1$ entradas por fila. En esta serie de rutas, el identificador de nodo y la llave se implementan como una secuencia de dígitos en base 2^b.

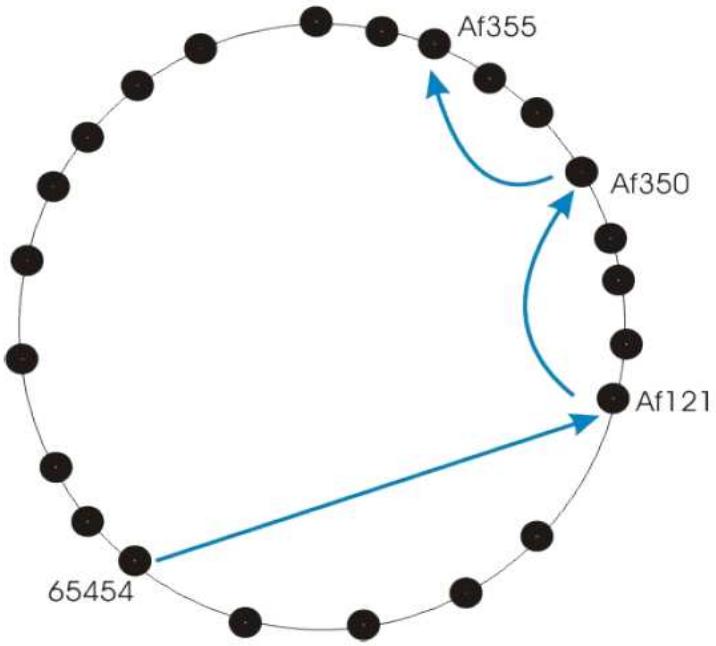

Fig. 2.3. Enrutado de mensajes

En la figura 2.3, se puede observar un ejemplo de enrutado para un mensaje destinado al nodo AF355 desde el nodo 65454. El nodo mira su tabla de rutas y se lo envía al nodo AF121, que tiene los mismos 2 dígitos en la primera parte del identificador, este a AF350 y así hasta llegar al nodo destino. De esta forma se consigue llegar a todas las partes del anillo sin necesidad de mantener rutas con todos los nodos del sistema ni de pasar por todos ellos.

Freepastry basa sus métodos de localización en la proximidad métrica. La proximidad métrica es un valor escalar que refleja la distancia entre un par de nodos, como pudiera hacer el RTT (Round Trip Time) del protocolo TCP. Esto, permite determinar la distancia entre un nodo y cualquier otro dada su IP.

2.3.4. Past

El aplicativo Past fue diseñado sobre FreePastry, con el fin de permitir el almacenamiento de ficheros de forma remota a gran escala. Este sistema pretende ofrecer la manera de crear un sistema rápido y sencillo que proporcione un sistema de ficheros con las propiedades de escalabilidad, balanceo, robustez y la alta disponibilidad que ofrece FreePastry.

2.3.4.1. Terminología y entidades en Past

Past, al trabajar sobre Freepastry, agrega una serie de entidades que no se encontraban establecidas hasta ahora. A continuación, se explican brevemente estas entidades, para facilitar el entendimiento de la labor desarrollada por Past en las redes distribuidas.

- **Storage:** Es el encargado de proporcionar un servicio local de almacenamiento. Este almacenamiento puede ser tanto en disco como en memoria.

- **MemoryStorage:** Es una implementación de Storage. Crea un almacenamiento en memoria. Esta clase no está especificada para crear puntos de almacenaje persistente en disco.

- **PersistentStorage:** A diferencia del anterior, implementa un punto de almacenaje persistente en disco.

- **Cache:** Espacio de memoria finito, donde se replican datos a los que se ha accedido con anterioridad y de manera frecuente, para proporcionar un tiempo medio de acceso a ellos considerablemente menor.

- **LRUCache:** La *Least-Recently-Used* Cache, mantiene un acceso rápido a los datos recientemente usados en menor medida.

- **StorageManager:** Es el Almacenamiento sujeto a la cache. Los objetos insertados o recuperados de memoria, pasan automáticamente, a formar parte de la cache para proporcionar mayor velocidad de acceso en futuros accesos.

2.3.4.2. Funcionamiento

Imaginemos que disponemos de un sistema donde está trabajando Past sobre FreePastry donde se desea publicar un fichero en el sistema. El sistema asignará, un identificador de 160 bits, el cual se obtiene mediante funciones de HASH.

Seguidamente, el fichero en cuestión es replicado hacia los N nodos más cercanos en la distribución lógica de anillo, tal como se puede observar en la figura 2.3. Estos nodos comparten ciertas similitudes en sus identificadores, pudiendo llegar a mostrar coincidencias hasta los 128 bits más significativos del fileId.

La forma en la que los identificadores están asignados, provoca que el hecho de replicar un fichero no produzca elevadas tasas de sobrecarga en la red. Es importante recordar, que el fichero estará replicado por el anillo FreePastry de manera uniforme en aquellos con identificador de nodo más similar, y por ello la tasa de sobrecarga disminuye.

Gracias a esta estructuración, el fichero estará disponible en cada uno de los nodos a los que se ha replicado y, siempre que uno de los N nodos que contiene una réplica se encuentre en el sistema, se tendrá acceso a dicho fichero.

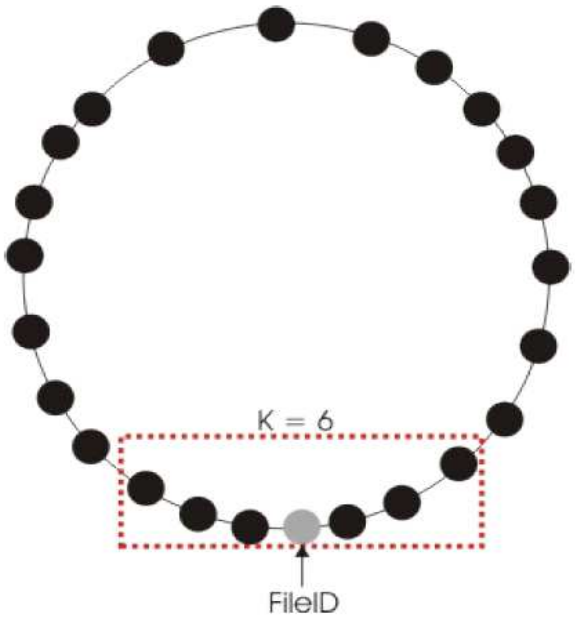

Fig. 2.3. Replicación Past.

En el caso de que se proceda a realizar una replicación a N nodos, existe la posibilidad de que alguno de estos nodos no disponga de la suficiente capacidad para almacenar el fichero. Para este tipo de caso, Past propone dos soluciones conocidas como Replica Diversion y File Diversion.

2.3.4.3. Replica Diversion

Replica Diversion es una solución propuesta por Past para combatir los problemas de espacio en la replicación de un fichero.

Imaginemos que se produce una replicación hacia diferentes nodos para que almacenen un fichero, y que el nodo X no puede almacenar dicho fichero por falta de capacidad de almacenamiento. X solicitará a otro nodo Y que almacene el fichero que él no puede guardar. El nodo Y se caracterizará por encontrarse fuera de los N nodos que originariamente fueron seleccionados para almacenar el fichero.

En caso de que el nodo Y almacenara el fichero en memoria, el nodo X almacena un puntero a dicho nodo. La función de este puntero es redireccionar las peticiones que él mismo reciba hacia el nodo que realmente tiene el fichero. Con el fin de mejorar la fiabilidad de este proceso, se añade otro puntero hacia el nodo Y. Este puntero estará almacenado en la tabla de los nodos más cercanos dentro del conjunto de los K nodos que contienen la réplica. En la tabla se hará referencia, como viene siendo hasta ahora, al identificador.

2.3.4.4. File Diversion

File Diversion consiste en cambiar la ubicación de almacenamiento de los ficheros. Se realiza un cálculo del fileId del fichero y se inserta en los K nodos más cercanos. Tras la sucesión de 3 intentos fallidos en llevar a cabo el File Diversion, se notifica el fallo en la realización de la acción mediante un mensaje de texto.

Cuando se produce una nueva llegada al anillo y éste sea situado por Id entre los K nodos, el nodo pasará a contener un puntero, con las mismas características que los explicados en Replica Diversion. Este puntero apuntará a uno de los nodos que contiene el fichero. Este efecto será de carácter temporal con el fin de poder proporcionar acceso al fichero mientras el fichero no sea añadido al nuevo nodo.

En el caso de que se produzca un abandono en el grupo de K elementos, se escoge un nodo que colinde con el grupo para añadirle el puntero hacia el fichero, y acabar quitándolo cuando este nodo contenga un versión completa del fichero.

Existe la posibilidad que de ante una caída de un nodo sea imposible añadir la información a un nuevo nodo. Ante este problema, se consulta a los dos nodos de los extremos de grupo de K elementos, si existe algún nodo fuera del grupo de K elementos capaz de alojar el fichero. En el caso de que se tenga conocimiento de un nodo capaz de almacenar el fichero, se crearían una serie de punteros enlazados hasta el nodo que contiene los datos.

De forma poco frecuente, se da la situación de que no sea posible garantizar las K copias requeridas para el sistema. En este caso, se espera que haya una nueva incorporación de alguno nuevo nodo o a que los ya existentes vuelvan a disponer de espacio de almacenamiento.

CAPÍTULO 3. DISEÑO DE LA APLICACIÓN

El desarrollo de una aplicación se inicia con una presentación lógica del diseño, especificando cada uno de los componentes que lo forman y las relaciones que existen entre ellos. La adecuación de un diseño a los requerimientos propuestos hace necesario una descripción de la aplicación, en la que se definirán las funcionalidades de sistema.

Seguidamente se mostrará un diagrama de clases, mediante el lenguaje de modelado UML, donde se visualizará la estructura de la aplicación y la organización.

Para finalizar, cada una de las clases será descrita brevemente y se definirán las relaciones que mantienen entre ellas.

3.1. Requisitos funcionales

La finalidad de la aplicación desarrollada, consiste en la creación de una red distribuida, aplicada a un entorno domótico simulado. La aplicación se centra en la autogestión de la iluminación de un recinto. Para ello se establecen dos tipos de elementos: sensores y elementos de control. A partir de esto se definen los requisitos funcionales:

- Acceso a la red: Cada nodo que acceda a la red, deberá notificarlo para recibir una tabla de rutas del estado actual del anillo.

- Control de Accesos y abandonos de la red: Se mantendrá un control sobre los accesos y abandonos a la red para que proporcionar un método de actualización a las tablas de rutas.

- Inserción o búsqueda de contenedores: En función del rol del nodo en la red, éste debe ser capaz de o bien realizar inserciones de contenedores de sus datos de localización o bien realizar la búsqueda de los contenedores de otros nodos.

- Envío de suscripciones a listas de notificación. Dependiendo de las tareas del nodo, éste deberá ser capaz de o bien realizar una petición de notificación ante cualquier modificación en la iluminación o bien identificar al suscriptor e inscribirlo en la lista de distribución de notificaciones.

- Notificación de eventos. Los nodos con lista de suscripción deberán realizar notificaciones a todos los integrantes de la lista informando variaciones lumínicas.

- Decisiones sobre medidas tomadas. Los nodos suscritos debe permanecer alerta ante cualquier notificación de variación lumínica y tomar las medidas adecuadas ante las variaciones tomadas.

3.2. Descripción del escenario y de la aplicación

Para poder cumplir con las características expuestas en el apartado anterior, los elementos de la red estarán divididos en dos grupos. Por un lado están los elementos establecidos para recoger datos de luminosidad, nodo con funciones de sensor en este caso, y otros preparados para interpretar esos datos y actuar en consecuencia, nodos de control.

Los Sensores, se encargarán de establecer las variaciones en la luminosidad del habitáculo. El elemento de control hará las funciones de interpretar los datos provenientes de los sensores y llevar a cabo las funciones adecuadas en cada momento.

Cada nodo sensor, realiza una publicación de datos en la red, que se verán replicados a N diferentes nodos. Esta publicación se realiza con el fin de anunciarse a los elementos de control. En la figura 3.1, se puede observar con más detalle los procesos que realizan los nodos e identificar los realizados por cada uno.

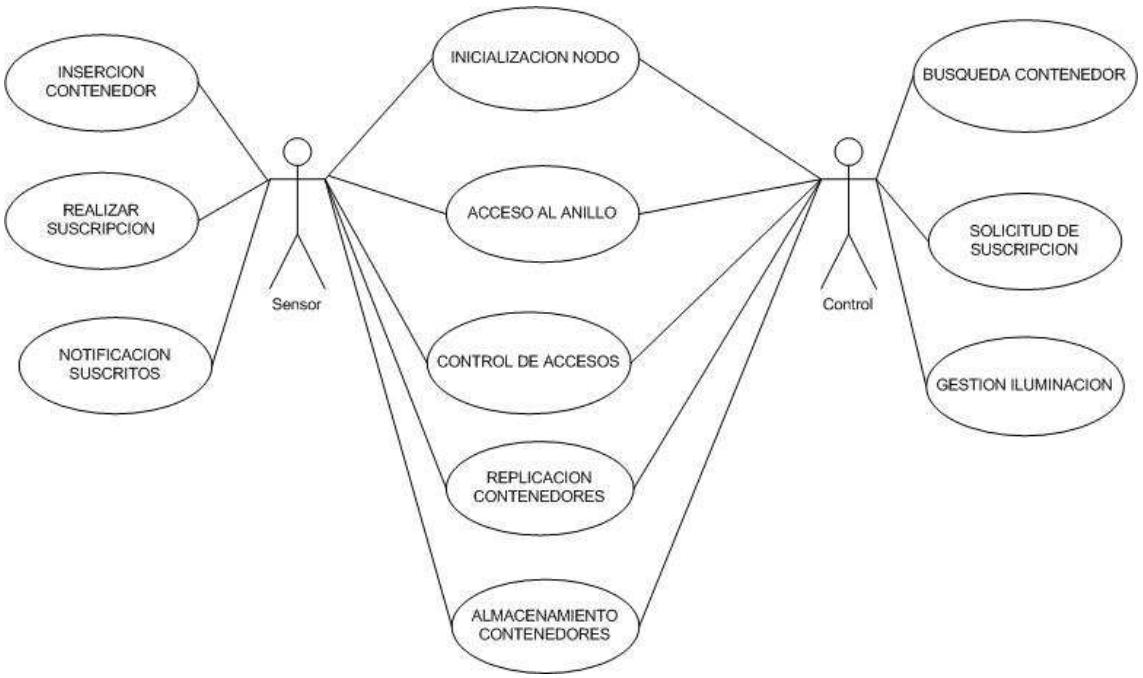

Fig. 3.1. Diagrama de Casos de Uso

En el momento en que los elementos de control pueden establecer contacto con los sensores, se suscriben a una lista en ellos. El objetivo de esta

suscripción es recibir las notificaciones referentes a las variaciones en la iluminación del recinto.

Por cada variación de intensidad que experimenta un sensor, se realiza una notificación a todos los elementos de control suscritos a su lista. Estas notificaciones permiten al controlador tomar decisiones, como aumentar la intensidad de la iluminación del recinto y encender o apagar la iluminación.

A continuación se explicará cada una de las partes que contienen los elementos de este montaje, para finalmente tener una visión más global y poder entrar en mayor detalle en la estructura.

3.2.1. Inicialización de nodo

Independientemente del rol que el nodo represente en la red, debe realizar un proceso de inicialización. Este proceso es idéntico para todos los nodos existentes en la red.

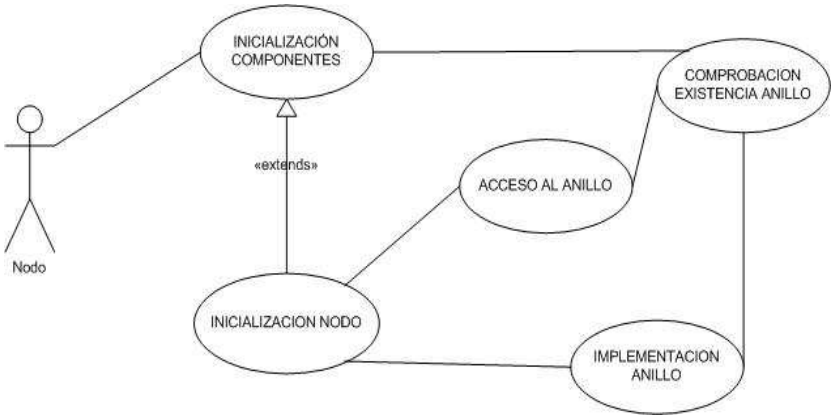

Fig. 3.2. Inicialización Nodo

Tal como se aprecia en la figura 3.2, primeramente suceden una serie de inicializaciones del sistema, básicas para el funcionamiento. Los elementos inicializados en este proceso son NodeIdFactory, PastryNodeFactory y NodeHandle. Estos elementos, definidos en el capítulo 2, se encargan de ofrecer servicios de generación de identificadores, inicialización de protocolos de transporte y tablas de rutas, y gestión de los protocolos de transporte para las conexiones de red.

Tras el proceso de inicialización, se comprueba si existe algún anillo que esté creado. En caso afirmativo, se establece conexión con el anillo y en caso negativo el host inicializa su propio anillo. Esta comprobación, se realiza a

través del componente de tipo NodeHandle, que contiene datos de protocolos de transporte y de red de la estación.

Tras el proceso de comprobación del anillo, se pone en ejecución el nodo. Es a partir de este momento, cuando todos los componentes están iniciados y se permite trabajar con tablas de rutas, de Hash, observadores, mensajería y demás opciones que presentan los nodos, en las que no entraremos en detalle.

Por último, se establece un objeto de tipo Listener para tener un control sobre los accesos y abandonos de la red por parte de cualquier nodo y para que la tabla de rutas sea debidamente controlada.

3.2.2. Creación de contenidos

La publicación de datos en un nodo, requiere que estos datos estén serializados. Intuitivamente, esta problemática surgida, se asemeja al caso expuesto en la figura 3.3. Cuando intentamos abrir un fichero del tipo PDF, con un editor como puede ser el caso del Microsoft Office Word, nos encontramos que el editor no puede interpretar los datos del fichero.

Fig. 3.3. Ejemplo Serialización.

En el caso de la aplicación del proyecto, los datos que un nodo publica para que sean accesibles a otros nodos, siguen una estructuración. Con la finalidad de que al recibir los datos desde otro nodo, sean interpretables y no confundidos con una serie de bits sin sentido para el receptor, se establecen los contenedores. Los contenedores son objetos definidos que establecen el formato de los datos publicados.

Para que el contenedor sea interpretable por cualquier nodo, se hace necesario que la definición de este contenedor conste en el nodo dispuesto a recibir los datos publicados.

Existen dos tipos de contenedores necesarios para el funcionamiento de la aplicación. El primero de ellos es el que contiene la información que se desea publicar, mientras que el segundo tiene un carácter más orientado a la gestión y a la búsqueda no directa de los datos publicados.

3.2.2.1. Contenedor de descripción

El contenedor de descripción se ha establecido para contener la definición del modelo y características del componente en la red, dada la orientación de la aplicación.

Este elemento, identificado en la aplicación como pContent, contiene también el identificador obtenido al aplicar una tabla de Hash al nombre de descripción del elemento en la red.

3.2.2.2. Contenedor de Id

Este contenedor establece los elementos necesarios para el funcionamiento de la red.

Cuando un contenedor de descripción es insertado en un nodo de la red, se crea una correspondencia en la tabla de Hash que vincula los datos insertados con un identificador. Este identificador suele establecerse mediante la aplicación de una tabla de hash al identificador de nodo.

La problemática que conlleva el uso de esta nueva inserción de datos, es que cada nodo puede tener más de un fichero insertado. Para ello este contenedor permite vincular un identificador de nodo con una tabla de identificadores, donde cada uno ellos hacen referencia a los diferentes datos insertados.

3.2.3. Replicación de contenedores

Previamente a la replicación de contenidos, y tras el establecimiento de conexión con el anillo por parte de los nodos, se necesitan una serie de elementos para la inserción de los datos. Estos elementos realizan las funciones de creación del punto de almacenaje, definición del tipo de almacenado y de gestor de directorios y replicaciones.

En este apartado es necesario tener en cuenta que tipo de hospedaje se le quiere dar a los ficheros, si de carácter persistente o en caché. También se deben fijar valores referentes a los tamaños que permitirán tanto el punto de almacenaje como el gestor para las replicaciones.

Para la inserción de contenedores en la red, se pone en práctica una técnica denominada *Continuation.* Esta técnica conlleva la creación de una nueva clase en la paso de parámetros de la función que inserta los datos. Esta práctica provoca que, mientras el proceso de inserción no haya finalizado, no continúe la ejecución, permitiendo que las condiciones sean las idóneas para este proceso.

3.2.4. Mensajes y notificación de eventos

La estructura de la aplicación hace necesario la implementación un método para la transmisión de notificaciones y para permitir el envío de mensajes.

Para ello, se ha definido una aplicación externa que ejecuta el nodo. Ésta desarrolla las funciones de enviar y recibir mensajes y/o notificaciones de eventos. Esta aplicación ha sido pensada para funcionar mediante un mecanismo de publicación-suscripción.

En la figura 3.4 se muestra un ejemplo de publicación-suscripción. Imaginemos un grupo de compradores interesados en un producto. Imaginemos también que es un producto muy novedoso que se agotará con suma rapidez y que la empresa de ventas ofrece un servicio para anunciar cuando dispone de stock.

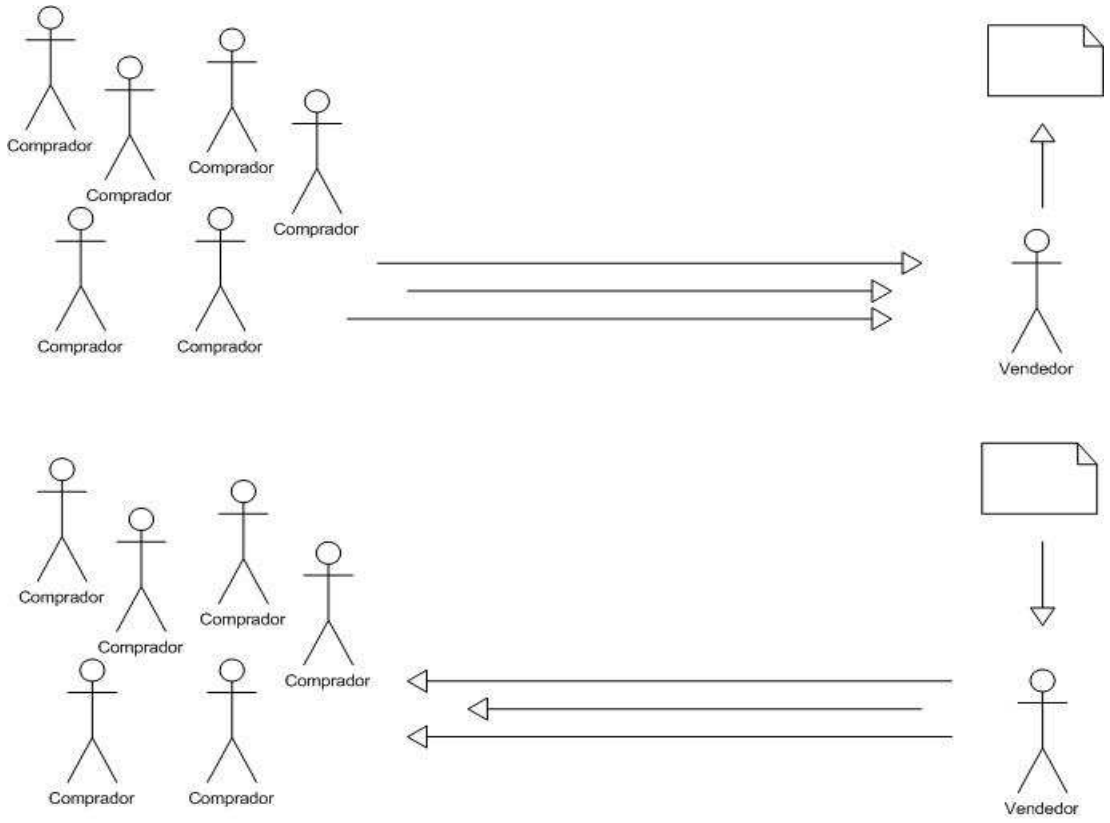

Fig. 3.4. Ejemplo notificación-suscripción

Para que los compradores sean informados de la existencia de nuevo stock, primeramente deberán ponerse en contacto con el vendedor y anunciarle que quieren ser informados en un fututo. Cuando el vendedor reciba stock, consultará su listado de clientes y realizará una notificación a cada uno de ellos.

Las entidades que controlan los términos de variación de luminosidad gestionan una lista de suscripciones similar a la gestionada por el vendedor de la fig. 3.4. En esta lista, se hallan los nodos que desean ser informados acerca de las variaciones.

Teniendo en cuenta que cada nodo en la red tiene un comportamiento diferente, dependiendo de si son suscriptores o centro de suscripciones, se ha definido la aplicación externa de transmisión.

En el caso de las entidades que gestionan las listas de suscripción, la aplicación está preparada para identificar si el mensaje recibido consiste en un mensaje de texto o consiste en solicitud de suscripción.

Por otro lado, las aplicaciones externas de las entidades que se suscriben, tienen implementada la transmisión de mensajes de suscripción y son capaces de recibir y distinguir mensajes de texto o de notificación de variaciones en la intensidad lumínica.

Para simplificar la especificación de los diferentes mensajes que circulan entre los nodos y permitir una fácil integración de nuevos eventos, se ha generado una jerarquía de mensajes.

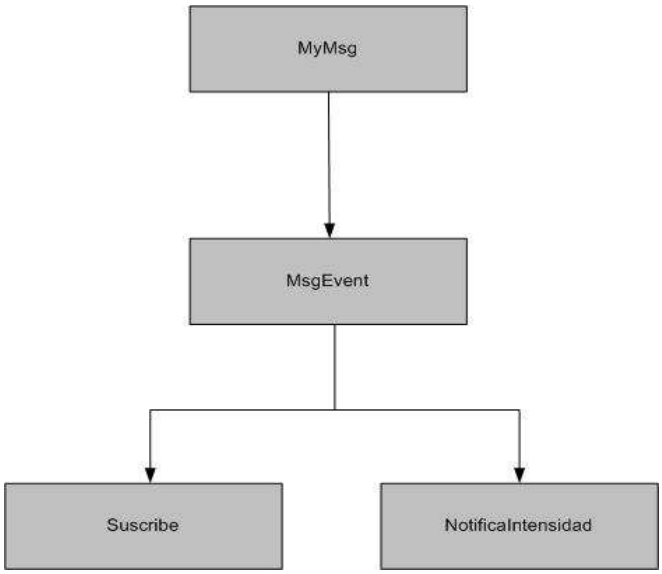

Fig. 3.5. Herencia Mensajes

Como se puede observar en la figura 3.5, la jerarquía o herencia entre los diferentes tipos de mensajes, permite ampliar el número de mensajes de notificación de eventos, con suma facilidad. En caso de ser necesario un nuevo tipo de mensaje, se debe tener en cuenta su función, la posición que ocuparía y de quien heredaría.

En los siguientes apartados, se analizan los mensajes establecidos, las características que tienen y el uso para el que han estado diseñados.

3.2.4.1. MyMsg

MyMsg es el primero patrón en el diagrama de herencia. Todos los mensajes diseñados, heredan de él. MyMsg se ha diseñado con una alta simplicidad, ya que, al heredar todos los demás mensajes de él, se debe garantizar que los campos entre los diferentes tipos sean de utilidad. Por ejemplo, un campo necesitado en un mensaje carácter específico, no sería apropiado ponerlo en MyMsg pues todos los demás lo heredarían y posiblemente no supondría una utilidad sino que provocaría una mayor carga de transmisión.

MyMsg únicamente presenta opciones para contener identificadores de origen y destino, y un mensaje de texto. MyMsg hereda de la opción propuesta por la librería CommonApi de Java, *Message*.

3.2.4.2. MsgEvent

La importancia de este mensaje reside en que todos los mensajes de notificación de eventos heredan de él y por tanto comparten su estructura. Simplemente fue diseñado para la herencia de los mensajes de notificación y no para ser usado en tareas de envío.

Este mensaje presenta las propiedades heredadas de MyMsg e incluye un campo para indicar el tipo de evento. Este campo facilita la interpretación al recibirse un mensaje del tipo evento y su tratamiento correspondiente.

3.2.4.3. Suscribe

Suscribe es el mensaje mediante el cual, los nodos Control que desean inscribirse a las notificaciones de un Sensor, solicitan la inscripción a estas listas de distribución de eventos. Éste hereda de MsgEvent por lo que incluye todos atributos de MsgEvent y MyMsg.

3.2.4.4. NotificaIntensidad

NotificaIntensidad es utilizado para que los sensores notifiquen a la lista de suscritos las variaciones que se puedan producir en la intensidad que captan. Para ello, éste incluye la definición de un atributo numérico, encargado de contener las variaciones, que se suma a los atributos heredados de MsgEvent y MyMsg.

3.3. Sensor

Llegados a este punto, una vez definidas todas las partes que forman el montaje, se dará una visión más específica de los diferentes roles que toman los nodos en la red.

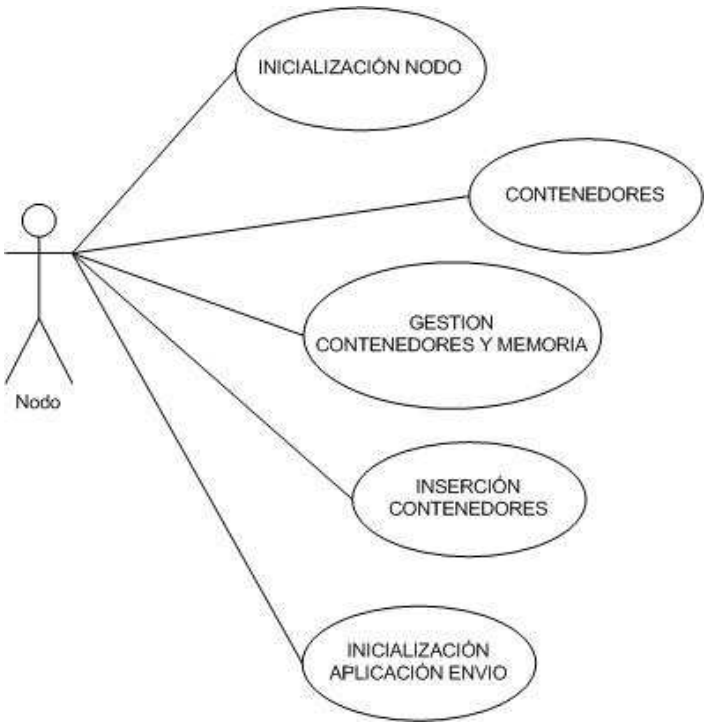

Fig. 3.6. Nodo Sensor.

En la figura 3.6, se puede observar la inicialización de un Sensor en la red. En primer lugar, se realizar la inicialización de nodo, tema que se abarcó con detalle en el apartado 3.1.

Cuando el nodo ha sido inicializado y conectado al anillo, ya estuviera establecido con anterioridad o sea de nueva acceso, se procede a la creación de los contenedores.

Tal como se define en el apartado 3.1.2, se establecen dos tipos de contenedores. El primer contenedor, almacena las características del elemento e identificadores y el segundo contiene una tabla donde se relaciona el identificador del nodo con todos los contenedores que publica el nodo.

Seguidamente, se inicializan los patrones de almacenamiento y el gestor de contenedores. Es en éste último donde se definen parámetros, como el punto de almacenamiento, la capacidad o el número de replicas. Este valor indica en qué cantidad de nodos en la que los contenedores que se desean replicar, serán insertados.

Tras el establecimiento de los contendores, cabe la posibilidad de que el sensor sea el primer elemento en la red. En ese caso, este elemento permanece a la espera hasta que existan el número de nodos necesarios para la replicación de los contenidos. En el caso de no ser el primer elemento en la red, y si ya existe el número de nodos adecuados para la replicación, se procede a ello.

Como último paso, se inicializa la aplicación de mensajería. Esta aplicación quedará en funcionamiento en un hilo de ejecución y será la que deberá interpretar los diferentes mensajes que se envíen a la red, tal y como se explica en el apartado 3.1.4.

3.4. Control

El nodo que ejerce de controlador en la red, presenta aspectos comunes con el sensor, pero su comportamiento en la red es totalmente diferenciado.

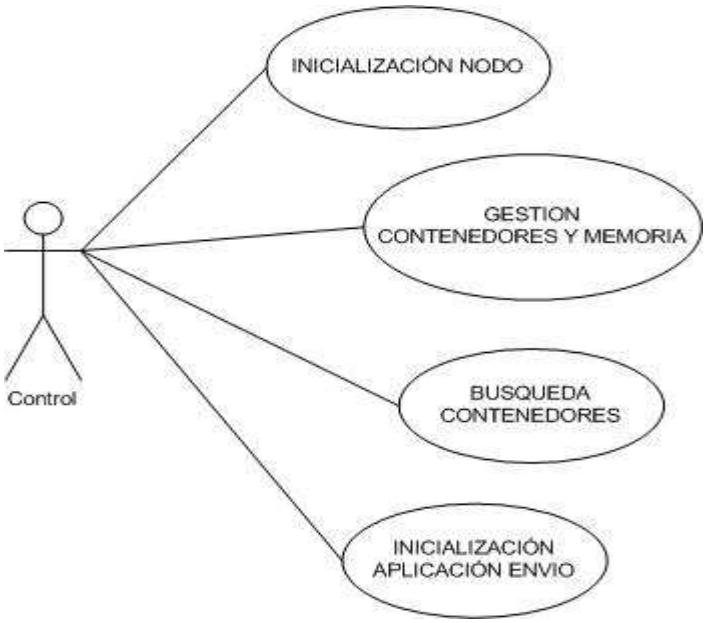

Fig. 3.7. Nodo Control

En la figura 3.7, se observan los puntos por los que un nodo de control debe pasar.

En primer lugar, se realizará la inicialización que, como ya se ha comentado, es un paso común para todos los nodos en la red.

Seguidamente, se debe establecer un punto de almacenaje y gestión de contenedores. Este paso es necesario porque, al realizarse la búsqueda del

contenedor, se necesita un punto donde poder depositar el elemento encontrado. Otra razón de esta necesidad, es que sin el gestor de contenedores no se podría realizar la búsqueda de elementos insertados en la red ya que, es a través de éste se realizan las búsquedas de contenedores.

La búsqueda de contenedores, en esta aplicación, adquiere un sentido meramente funcional. El hecho que se desee que los nodos interactúen entre ellos de manera automática, requiere que se inserten los contenidos en la red y que el nodo realice la búsqueda. Posteriormente, el nodo control descargará el contenedor del nodo al que se desea suscribir, de donde extrae los datos necesarios para establecer comunicación con él.

Como último paso se activa la aplicación externa de mensajería. Ésta se encargará de suscribir al nodo control en la lista de distribución de un nodo sensor y permanecerá en ejecución para recibir las notificaciones acerca de las notificaciones en la variación de la luminosidad.

3.5. Diagrama de clases

El diagrama de clases consiste en representar el diseño de la aplicación mediante la estructura de las clases utilizadas para su realización.

En la figura 3.8, se muestra la jerarquía de las clases. Para cada una de ellas, se especifican los atributos y métodos correspondientes.

La clase principal, la cual se encuentra al nivel más alto, es la que engloba el conjunto de objetos utilizados para la aplicación. Ésta es la que interacciona con la red y muestra los resultados en pantalla para que el usuario puede interpretar.

En el diagrama de clases, las flechas continuas indican qué clases utilizan a otras clases de la aplicación. Las discontinuas indican qué clases extienden de otras clases definidas.

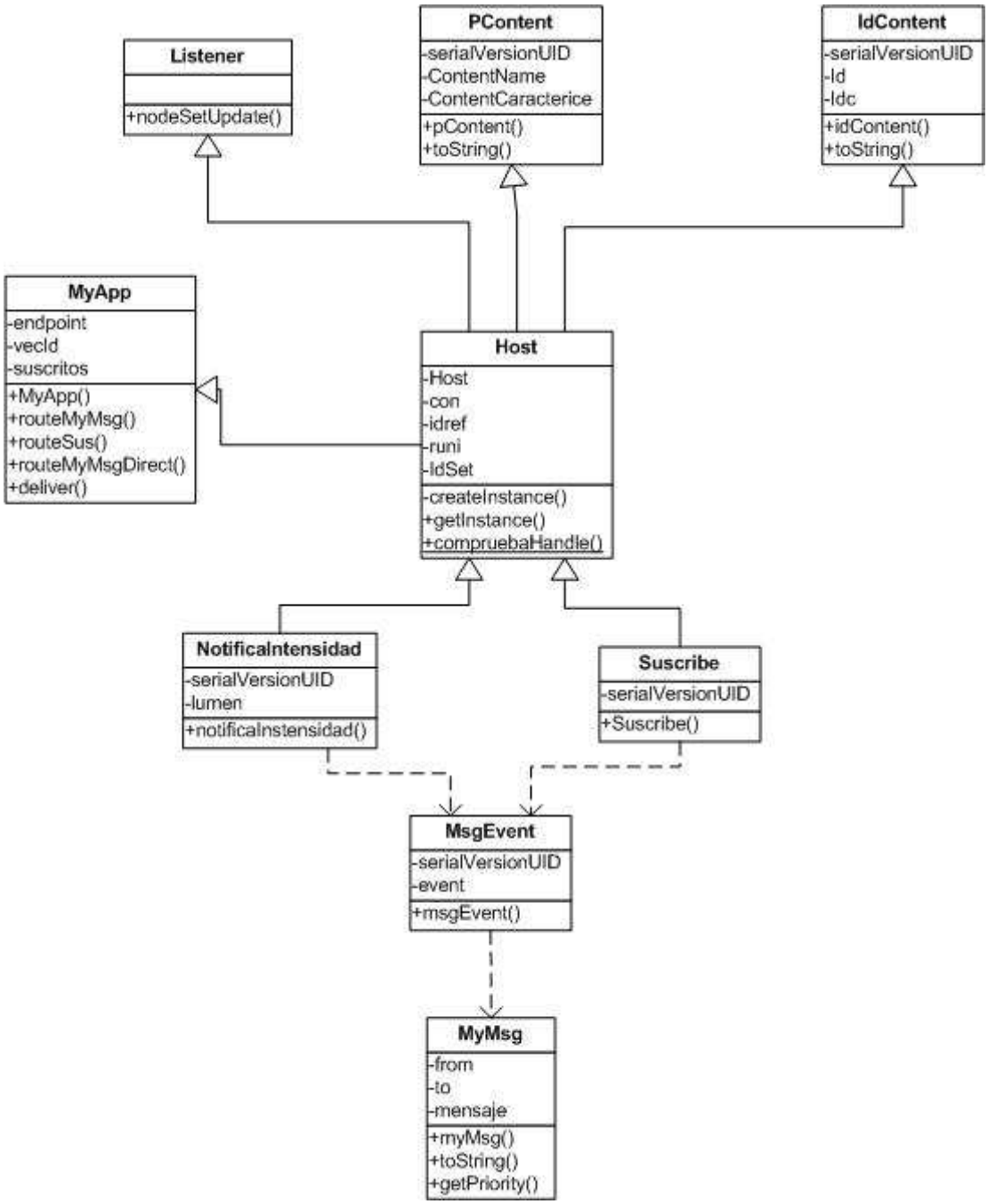

Fig. 3.8. Diagrama de Clases

3.6. Descripción de clases

En este apartado se procede a describir brevemente cada una de las clases de la aplicación i las funcionalidades que desempeñan.

- **Clase Host:** Esta clase es la representación de un peer. Contiene todos los atributos básicos que lo caracterizan y realiza las funciones de

detección de accesos de nuevos peers a la red, inseción contenedores, realizar búsquedas de contenedores, envío de suscripciones o notificaciones y recepción de suscripciones o notificaciones.

- **Clase Listener:** Esta clase permite que el peer pueda detectar el acceso o abandono de un peer al anillo. Para ello, implementa la interface propuesta por FreePastry, NodeSetListener, que nos permite trabajar con listeners para las tablas de rutas. La clase Listener actúa como un proceso independiente que se mantiene en ejecución a la espera de notificaciones de acceso o abandono de la red. Cuando existe una notificación de este tipo, cada peer actualiza su tabla de rutas.

- **Clase PContent:** Permite, a cada peer que lo contenga, serializar los datos insertados en el anillo. En el caso concreto del diseño propuesto para esta aplicación, contiene el nombre del dispositivo y una breve definición de su función. Cada peer que deba acceder a los datos del contenedor, deberá tener acceso a esta clase para poder interpretar la serialización del contenedor PContent. Esta clase extiende de la clase propuesta de FreePastry, ContentHashPastContent.

- **Clase IdContent:** Clase que extiende también de ContentHashContent. Utilizada para realizar un listado de los diferentes contenedores que posee un peer. En él, se relacionan los identificadores de los contenedores de un nodo, con el identificador del peer. Su labor es meramente funcional, ya que otorga a los peers la posibilidad de realizar las búsquedas a través de los identificadores de nodos y permitir la interpretación de estos contenedores.

- **Clase MyApp:** Esta clase permite la transmisión y la recepción de los mensajes referentes a los eventos que conciernen a la suscripción en otros peers o la notificación de eventos de intensidad lumínica. Esta clase actúa como proceso independiente, manteniéndose en ejecución para la recepción o transmisión de los diferentes tipos de mensajes.

- **Clase MyMsg:** Clase que hereda de la propuesta realizada por la api de Java, Message. Define los atributos básicos para la transmisión y/o recepción de un mensaje, indiferentemente del tipo que sean. En esta clase, se define una serie de atributos, que son heredados genéricos preestablecidos para su uso en otras clases que hereden de ésta.

- **Clase MsgEvent:** Clase con herencia de MyMsg. Define los campos básicos comunes que todo mensaje concerniente a un Evento debe tener.

- **Clase Suscribe:** Hereda de MsgEvent y está designado para realizar las suscripciones a los diferentes nodos.

- **Clase NotificaIntensidad:** Clase con herencia de MsgEvent. Presenta, incluyendo toda lo heredado de MsgEvent, un campo de indicar variaciones de intensidad lumínica.

CAPÍTULO 4. IMPLENTACIÓN Y PRUEBAS

En este capítulo se explican los diferentes aspectos relacionados con la aplicación diseñada anteriormente, como las tecnologías utilizadas o la interacción de la aplicación. Finalmente, se incluyen las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación y la manera en que se ha estructurado este proceso de pruebas.

4.1. Tecnologías y herramientas utilizadas

Las herramientas necesarias para la implementación de la aplicación son las siguientes:

- **Java Runtime Environment 1.5.0_04.** Versión de Java 2 que contiene las librerías y la máquina virtual de java, necesarias para la ejecución de la aplicación.

- **Freepastry 2.0.** Librería que contiene las especificaciones referentes a la tecnología P2P utilizada.

- **Xmlpull_1_1_3_4c.** Librería que contiene los métodos de parseo a XML. Esta librería, utilizada por Freepastry, es del tipo pull-parsing. Funciona como un iterador de los elementos XML y es la aplicación la que solicita los eventos, estando la información implícita en el propio código de la aplicación.

- **Xpp3-1.1.3.4.D.** Librería también utilizada para el parseo a XML. Es un requisito de Freepastry.

4.2. Interacción entre aplicación y usuario

Como se explica en el proceso de diseño, esta aplicación fue pensada como una solución de autogestión y autoconfiguración. Debido a que una interfaz gráfica implica un control directo del usuario se ha prescindido de establecer una y se ha procedido a realizar todas las pruebas mediante la consola.

En el caso de la intervención del usuario para modificar el estado de la iluminación en el recinto, sería pertinente disponer de una interfaz gráfica para facilitar estas tareas.

La aplicación requiere una serie de datos para su correcta ejecución. Son necesarios el paso como argumentos, el puerto del socket del nodo local, la dirección IP del nodo que ha generado el anillo y el puerto de este mismo nodo.

4.3. Pruebas realizadas

Para establecer un proceso de ejecución eficiente, el montaje de la aplicación fue estructurado en diferentes partes. De esta manera, cada parte finalizada es probada individualmente asegurando su correcto funcionamiento tras diferentes ensayos.

Cada uno de los siguientes apartados se corresponde con una de las fragmentaciones realizadas en el montaje de la aplicación. En estos apartados se explica brevemente los ensayos realizados a cada uno de ellos.

Principalmente se ha comprobado el buen funcionamiento de los métodos contenidos en cada clase. También se ha comprobado la correcta interacción entre los objetos que componen el sistema.

Tras las pruebas individuales de cada parte de la aplicación, se han realizado una serie de comprobaciones de manera más global. Se han realizado también una serie de pruebas con un sniffer de paquetes para observar el tráfico generado y los retardos.

4.3.1. Establecimiento del anillo Freepastry

Esta primera prueba tenía como objetivo comprobar dos aspectos básicos pero importantes para el funcionamiento de la aplicación. El primero de ellos era comprobar que la inicialización de los nodos era correcta y, en caso de no existir ningún anillo lógico creado previamente, se estableciera un nodo como responsable de proporcionar acceso al anillo.

También se realizó la comprobación del buen funcionamiento del Listener responsable de monitorizar la entrada o salida de nodos en la red.

En la figura 4.1, de la página siguiente, se muestran los resultados de estas las pruebas comentadas.

En el primer recuadro, se muestra el proceso de inicialización. Se observa en el resultado mostrado que el nodo es el primero en establecer conexión y que, por tanto, no encontrará ningún anillo creado anteriormente. Por este motivo aparece un rechazo de conexión. Es en este momento cuando el nodo interpreta que no existe un anillo previo y establece su propio anillo. Este anillo será gestionado por el componente del nodo, denominado BootStrapNode.

Como garantía de éxito en estas primeras comprobaciones, una vez que el nodo ha sido creado se observa que, tras el mensaje de inicialización satisfactoria del nodo, se retornan los datos de conexión del nodo.

```
:rice.pastry.socket:1233607792834:Error connecting to address /147.83.118.124:19001: java.net.ConnectException: Connection refused: no further information
:rice.pastry.socket:1233607792865:No bootstrap node provided, starting a new ring binding to address DAVID/147.83.118.124:19002...

  El nuevo nodo ha sido creado satisfactoriamente

SocketNodeHandle (<0x727309..>/DAVID/147.83.118.124:19002 [90156383303981098691])

Acceso de nuevo nodo a la red
```

```
  El nuevo nodo ha sido creado satisfactoriamente

SocketNodeHandle (<0x97AB7C..>/DAVID/147.83.118.124:19001 [340400341552080355O])

Acceso de nuevo nodo a la red

Acceso de nuevo nodo a la red

Nodo con Id:<0xA6CB69..>desconectado del anillo
```

Fig. 4.1. Accesos e Inicialización de nodos y anillo

En el segundo recuadro se pueden observar, algún patrón en común con el primero, como por ejemplo el establecimiento del retorno del SocketNodeHandle. Pero también se aprecian algunas diferencias que pasamos a comentar.

En primer lugar, nótese que en el recuadro derecho de la figura 4.1, el proceso de inicialización no muestra el rechazo de conexión que aparece en la primera figura. Esto es debido a que el anillo ya está creado y únicamente se realiza la petición de acceso, para seguidamente devolver, como en el caso anterior, los datos de configuración de red.

Obsérvese también que en ambos aparecen una serie de mensajes referentes al acceso de nuevos nodos en la red. En el caso de la figura derecha se aprecia también un mensaje en el que se notifica el abandono de un nodo de la red. Esta serie de mensajes son proporcionados por la función que desempeña el Listener de la tabla de rutas. Mediante este proceso se mantiene la tabla de rutas actualizada de los posibles accesos o abandonos que se dan en la red.

4.3.2. Past, la replicación y la búsqueda de datos

Esta serie de pruebas sucede en el punto en el que los nodos pueden establecer la conexión. A continuación, y tras encontrar las condiciones idóneas, se procede a almacenar los datos de manera local y a realizar la réplica de datos a la red.

La inserción y la replicación de datos no tiene sentido sin que exista una entidad en la red que tenga la necesidad de acceder a estos datos. Es por este motivo que se mostraran, en este apartado, las pruebas relacionadas con la búsqueda de datos insertados en la red.

Con el fin de poder observar estos factores y su evolución, se configuran diferentes nodos con diferentes funciones específicas. Existirá, por tanto, un nodo que realizará la inserción de datos en local. La réplica de ficheros conlleva la existencia de un factor mínimo de nodos en la red, que será fijado en 2. Esto significa, que hasta que no existan 2 nodos más, aparte del nodo publicador, no se procederá a replicar. Cuando estas condiciones se cumplan, se procederá a insertar los datos en la red.

Para la réplica, se definen unos nodos que como única función tendrán que recibir las replicas emitidas desde el nodo publicador, para almacenarlas localmente.

Como último paso se establecerá un nodo programado para que, en el momento en el que el número de nodos necesario para la réplica se ajuste y la réplica haya sido cursada exitosamente, éste nodo proceda a realizar la búsqueda de elementos para posteriormente almacenarlos en local.

En la figura 4.2, se ilustra la inserción de datos. Obsérvese que se realizan dos inserciones tras el establecimiento de los nodos N necesarios para la replicación, mostrándose ambas por pantalla una vez realizadas. La primera inserción, define el tipo de equipo y sus características, mientras que la segunda es meramente funcional y define un listado de Id relacionados con el identificador del nodo.

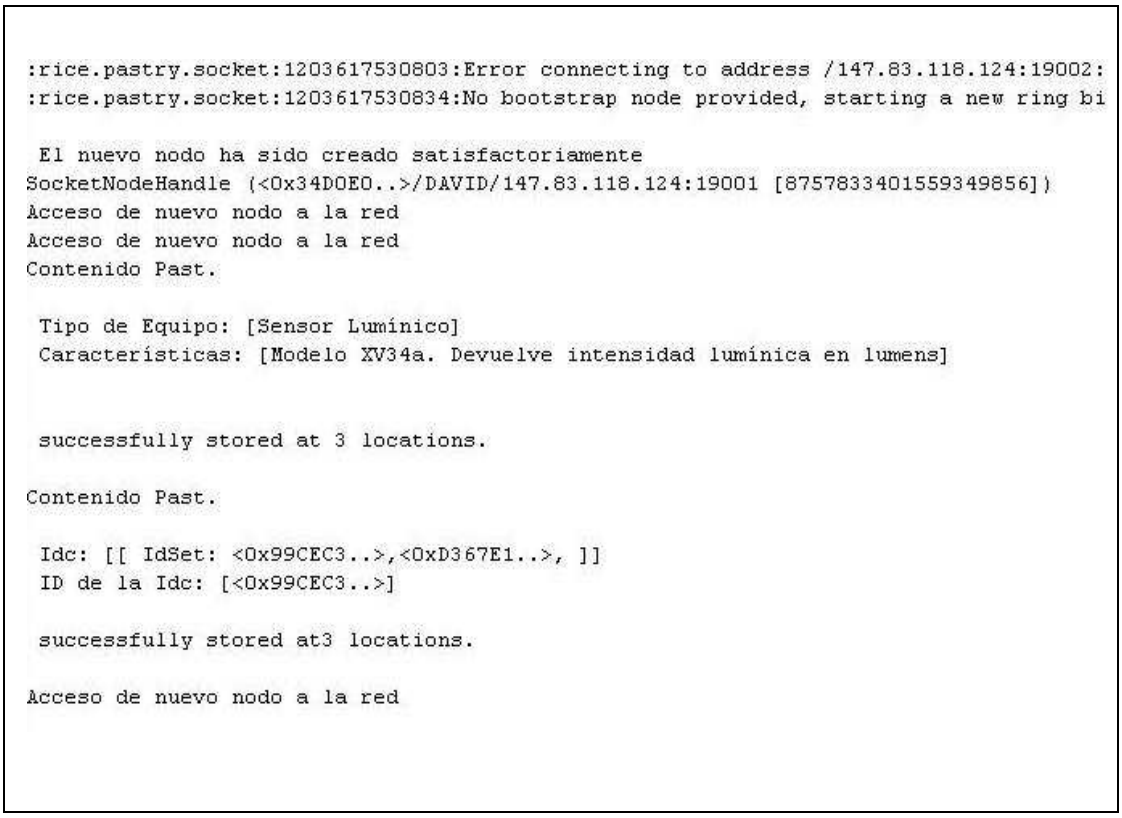

Fig. 4.2. Replicación

Tras la replicación, un cuarto nodo realiza la búsqueda del contenedor de su interés, tal como se aprecia en la figura 4.3.

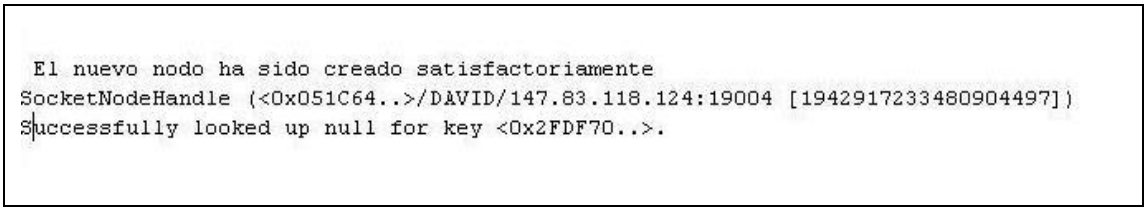

Fig. 4.3. Búsqueda

La realización de estas pruebas se ha ejecutado mediante diferentes máquinas virtuales en ejecución sobre el mismo equipo. Esto facilita que podamos observar en la figura 4.4, como tanto los datos insertados, replicados a los

otros nodos, como los buscados finalmente, han sido almacenados correctamente. Nótese que en el punto de almacenamiento, aparecen 4 directorios. Local-XXXXX, se corresponde con el directorio que utiliza el nodo que realiza la inserción. Los directorios replicaXXXX, son los creados una vez se han enviado los contendores a los otros nodos en la red. Por último el directorio busquedaXX, que es el resultado de realizar la búsqueda desde el cuarto nodo.

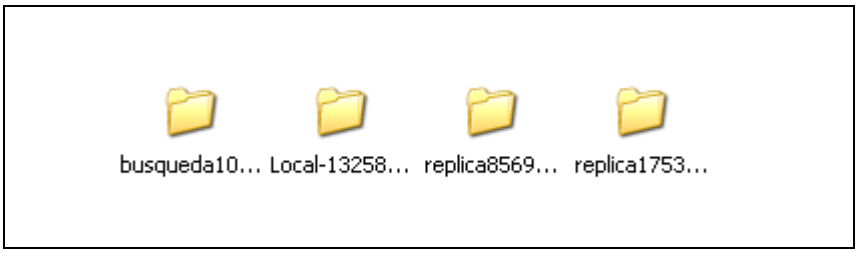

Fig. 4.4. Almacenamiento de los contenedores

Todos ellos contienen en su interior los ficheros insertados, replicados o buscados, con el matiz de que sin la serialización necesaria son ininteligibles.

4.3.3. Envío de notificaciones y mensajes

En este apartado de pruebas, se desea verificar el correcto funcionamiento en la aplicación de envío de mensajes y la compatibilidad de ésta con el resto de procesos probados en los nodos.

Para ello, se establece una red de iguales características a la probada en el apartado 4.3.2, con la diferencia de que, tanto el nodo sensor como el nodo control dispondrán de la aplicación de envío.

En el caso de la aplicación del nodo control, ésta efectúa el envío de un mensaje de suscripción, una vez ha localizado al nodo sensor. Tras esto quedará a la espera de posibles mensajes notificando variaciones lumínicas. En la siguiente imagen, se puede apreciar estos procesos comentados.

```
 El nuevo nodo ha sido creado satisfactoriamente
SocketNodeHandle (<0xF163F5..>/DAVID/147.83.118.124:19004 [2506933086279572977])
Successfully looked up null for key <0x2FDF70..>.

MyApp <0xF163F5..> sending to [SNH: <0xC7F601..>//147.83.118.124:19001 [-6273726173319920877]]
MyApp <0xF163F5..> received MyMsg from <0xC7F601..> to <0xF163F5..>
Recibida Intensidad Inicial de 3. Se procede a activar iluminación
```

Fig. 4.5. Mensajes Nodo control

La aplicación del sensor, en cambio, está configurada para recibir notificaciones y agregar cada solicitante a una lista para que, cuando se produzca cualquier variación en la iluminación, se proceda a enviar una respuesta indicando este nuevo estado.

```
:rice.pastry.socket:1203685848218:Error connecting to address /147.83.118.124:19002:
:rice.pastry.socket:1203685848265:No bootstrap node provided, starting a new ring bi

 El nuevo nodo ha sido creado satisfactoriamente
SocketNodeHandle (<0xC7F601..>/DAVID/147.83.118.124:19001 [-6273726173319920877])
Acceso de nuevo nodo a la red
Acceso de nuevo nodo a la red
Contenido Past.

 Elemento: [Sensor Luminico]
 Caracteriticas: [Indica intensidad luminica en lumens]

 successfully stored at 3 locations.

Contenido Past.

 Idc: [[ IdSet: <0x272B9F..>,<0x886B45..>, ]]
 ID de la Idc: [<0x272B9F..>]

 successfully stored at3 locations.

Acceso de nuevo nodo a la red
MyApp <0xC7F601..> received MyMsg from <0xF163F5..> to <0xC7F601..>

Mensaje de suscripcion:MyMsg from <0xF163F5..> to <0xC7F601..>del Id<0xC7F601..>
MyApp <0xC7F601..> enviando intensidad lumínica inicial a :<0xF163F5..>
```

Fig. 4.6. Mensajes Nodo Sensor.

En la figura 4.6, podemos observar cómo, tras la inserción de los contenedores de datos, aparece un nuevo mensaje, que es catalogado como mensaje de suscripción. Cualquier nodo de nueva suscripción recibirá una notificación del estado de la intensidad lumínica actual.

4.3.4. Varios nodos

Llegados a este punto, y comprobado el funcionamiento en una situación con el mínimo número de nodos, se plantea realizar la prueba de poner en interacción a más de dos nodos. Para ello, se emplean un total de siete nodos, cuatro sensores y tres elementos de control.

En la figura 4.7, encontramos los procesos de comunicación obtenidos en uno de los sensores. En él se pueden apreciar como van accediendo diferentes nodos, como se reciben las solicitudes de inscripción y como se les da respuesta a estas mismas.

```
:rice.pastry.socket:1203764893281:Error connecting to address /192.168.1.3:19010:
:rice.pastry.socket:1203764893546:No bootstrap node provided, starting a new ring

 El nuevo nodo ha sido creado satisfactoriamente
SocketNodeHandle (<0x1F5574..>/DAVID/192.168.1.3:19001 [-1176188723743378004])
Acceso de nuevo nodo a la red
Acceso de nuevo nodo a la red
Contenido Past.

 Elemento: [Sensor Luminico 1]
 Caracteríticas: [Indica intensidad luminica en lumens]

 successfully stored at 3 locations.

Contenido Past.

 Idc: [[ IdSet: <0x0B42C6..>,<0x93A164..>, ]]
 ID de la Idc: [<0x0B42C6..>]

 successfully stored at3 locations.

Acceso de nuevo nodo a la red
Acceso de nuevo nodo a la red
MyApp <0x1F5574..> received MyMsg from <0x7A641B..> to <0x1F5574..>

Mensaje de suscripcion:MyMsg from <0x7A641B..> to <0x1F5574..>del Id<0x1F5574..>
MyApp <0x1F5574..> enviando intensidad lumínica inicial a :<0x7A641B..>
MyApp <0x1F5574..> received MyMsg from <0x4E8763..> to <0x1F5574..>

Mensaje de suscripcion:MyMsg from <0x4E8763..> to <0x1F5574..>del Id<0x1F5574..>
MyApp <0x1F5574..> enviando intensidad lumínica inicial a :<0x4E8763..>
Acceso de nuevo nodo a la red
MyApp <0x1F5574..> received MyMsg from <0xDCFBB8..> to <0x1F5574..>

Mensaje de suscripcion:MyMsg from <0xDCFBB8..> to <0x1F5574..>del Id<0x1F5574..>
MyApp <0x1F5574..> enviando intensidad lumínica inicial a :<0xDCFBB8..>
Acceso de nuevo nodo a la red
MyApp <0x1F5574..> received MyMsg from <0x23FD9A..> to <0x1F5574..>

Mensaje de suscripcion:MyMsg from <0x23FD9A..> to <0x1F5574..>del Id<0x1F5574..>
MyApp <0x1F5574..> enviando intensidad lumínica inicial a :<0x23FD9A..>
```

Fig. 4.7. Resultados nodo sensor

En la figura 4.8, se muestran los mensajes resultantes de la comunicación de uno de los nodos de control. Obsérvese que éste, recibe diferentes mensajes de notificación inicial, uno por cada sensor, y que anteriormente a activar el elemento de iluminación, comprueba su estado.

```
 El nuevo nodo ha sido creado satisfactoriamente
SocketNodeHandle (<0x33096F..>/DAVID/192.168.1.3:19007 [-6207142255836677594])
Successfully looked up Contenido Past.

 Elemento: [Sensor Luminico 1]
 Caracteríticas: [Indica intensidad luminica en lumens] for key <0x93A164..>.
MyApp <0x33096F..> received MyMsg from <0x7EE11D..> to <0x33096F..>
MyApp <0x33096F..> received MyMsg from <0x24E755..> to <0x33096F..>
MyApp <0x33096F..> sending to [SNH: <0x645762..>//192.168.1.3:19004 [8127935569346713257]]
MyApp <0x33096F..> received MyMsg from <0x645762..> to <0x33096F..>
Recibida Intensidad Inicial de 3. Se procede a activar iluminación
Recibida Intensidad Inicial de 3. La iluminación ya se encuentra conectada
MyApp <0x33096F..> sending to [SNH: <0x7EE11D..>//192.168.1.3:19003 [-2104204548319967462]]
MyApp <0x33096F..> sending to [SNH: <0x88930B..>//192.168.1.3:19001 [6751669809737269934]]
MyApp <0x33096F..> received MyMsg from <0x88930B..> to <0x33096F..>
Recibida Intensidad Inicial de 3. La iluminación ya se encuentra conectada
MyApp <0x33096F..> sending to [SNH: <0x91E553..>//192.168.1.3:19002 [3546333437911155210]]
MyApp <0x33096F..> received MyMsg from <0x91E553..> to <0x33096F..>
Recibida Intensidad Inicial de 3. La iluminación ya se encuentra conectada
MyApp <0x33096F..> sending to [SNH: <0xADE31D..>//192.168.1.3:19005 [791807471476712596]]
MyApp <0x33096F..> sending to [SNH: <0x24E755..>//192.168.1.3:19006 [6255892780891610843]]
```

Fig. 4.8. Resultados nodo control

Gracias a estas pruebas realizadas podemos concluir que el comportamiento de la aplicación es correcto en todos los ámbitos propuestos en las diferentes pruebas realizadas, ya que en todas ellas se han alcanzado los requisitos demandados.

CAPÍTULO 5. BALANCES Y CONCLUSIONES

Una vez finalizada la implementación del sistema, es necesario analizar los resultados obtenidos y extraer de ellos una serie de conclusiones.

En este capítulo, se valoran los puntos del proyecto en los que se ha invertido un tiempo mayor. Seguidamente, se comprueba el cumplimiento de los objetivos marcados inicialmente. Se plantean también, posibles mejoras y ampliaciones futuras y se exponen las conclusiones personales que se han extraído después de realizar este proyecto.

5.1. Comprobación de la planificación inicial

El desarrollo del proyecto ha sido muy positivo, ya que se han alcanzado todos los objetivos marcados inicialmente, obteniendo los resultados esperados.

Durante el desarrollo de la aplicación se ha producido un desvío en la planificación inicial. En esta primera planificación, se pretendía que sobre los elementos de control el usuario pudiera ejercer un control mediante un servicio de webservice.

Como solución al servicio webservice se había escogido, Axis2 de Apache. Este servicio aparece con la necesidad de crear un canal interoperable entre dos máquinas conectadas a una misma red. Para ello, este modelo ofrece la posibilidad de encapsular diferentes tipos de lenguajes sobre http, con el fin de transmitir diferentes conjuntos de datos, sin ser una característica a tener en cuenta, el sistema operativo en que trabajan las diferentes máquinas de la red o en que lenguaje ha sido programada la aplicación.

Axis 2, soporta diferentes protocolos como WDSL y SOAP principalmente, mientras que los transportes, se pueden realizar mediante protocolos como HTTP, SMTP JMS y TCP. Por otro lado permite también el uso de Beans, ya sean propietarios de apache como puede ser el caso de Axis Data Binding o XMLBeans.

Mediante Axis 2, se pretendía que el usuario pudiera acceder a los datos de diferentes elementos de control i poder encender o apagar la iluminación.

El giro sufrido en el proyecto, es debido al desarrollo en la aplicación de envío que conllevó una alta carga temporal. No obstante, se concluyó que el control de usuario, se podía realizar también mediante la mensajería establecida entre nodos, asignando para el usuario un nuevo tipo de mensajes que fuera interpretable por los nodos de control.

Un factor positivo es el hecho de que tanto Java como Freepastry no eran herramientas con las que hubiera trabajado habitualmente y de las que he obtenido conocimientos muy valiosos para futuros proyectos.

5.2. Objetivos cumplidos

El propósito general del proyecto consistía en familiarizarse con la tecnología Freepastry y desarrollar una aplicación donde pudiera ser utilizada y donde se pudiera comprobar su correcto funcionamiento. Finalizado el proyecto, se puede afirmar que este propósito ha sido cumplido satisfactoriamente.

Analizando los objetivos iniciales del proyecto, se puede comprobar, independientemente de las modificaciones en el planteamiento, que se han obtenido en su totalidad. Esta memoria se estructura en función de los objetivos, dedicando un capítulo a cada uno de estos. Haciendo este seguimiento, se puede corroborar que inicialmente, se ha realizado un estudio de las tecnologías utilizadas seguido de un análisis específico de la tecnología Freepastry.

La aplicación diseñada, utilizando la metodología UML, se ha podido implementar consiguiendo los requerimientos iniciales. Desde un principio, se había previsto el uso de la tecnología webservices para la gestión del usuario. Aunque sobre esta tecnología se había realizado un estudio previo y se había puesto en práctica para comprobar su funcionamiento, se descartó su utilización por las facilidades que proporcionaba la aplicación de envío de notificaciones y que mediante ella el usuario podría ejecutar también las labores para las que se iba a disponer de la tecnología webservices.

Finalmente, se han realizado las pruebas adecuadas para comprobar que la aplicación funcionara de la manera esperada y, de esta manera, dar por concluido el proyecto.

El código fuente de la aplicación, en formato compilable, se encuentra en el CD de la versión digital del proyecto, y se puede comprobar cómo cumple los requisitos planteados en la etapa de diseño.

5.3. Mejoras y ampliaciones futuras

5.3.1. Posibles mejoras

Tras el desarrollo de la aplicación, se ha concluido que el diseño inicialmente propuesto es totalmente óptimo y aunque la aplicación realiza todas las funcionalidades requeridas inicialmente, está muy limitado el hecho de implementarlas de con sencillez.

En el diseño realizado finalmente, toda funcionalidad se controla en mayor o menor medida desde la clase host, hecho que conlleva una cierta complejidad. Por tanto, realizar un cambio en la estructura de las clases, evitando la alta

dependencia de la clase host, sería conveniente para que las posibles ampliaciones fueran más sencillas.

5.3.2. Ampliaciones futuras

Aunque los objetivos propuestos han sido cumplidos, la aplicación sólo ofrece unas funcionalidades muy básicas, teniendo en cuenta todas las posibilidades de que ofrece Freepastry. En este apartado por tanto, se describen posibles ampliaciones futuras para la aplicación desarrollada.

Past genera una serie de mensajes entre los nodos que están replicando los datos en la red. Desde esta aplicación no se está realizando ningún tipo de control sobre estos mensajes y, por otro lado, tampoco se ha realizado ningún tipo de estudio al respecto.

Una de las posibles ampliaciones futuras, es extender la aplicación a todo los componentes de un ambiente domótico. Una red domótica no únicamente está compuesta de elementos de iluminación, sino que también se encuentran elementos de temperatura, de alarmas, electrodomésticos, entre otros. Esto no sería muy costoso porque en un ámbito simulado, como lo es el propuesto aquí, únicamente añadiendo unas funcionalidades en la aplicación de envío, nuevos mensajes de notificaciones y que accedieran a la red P2P, sería factible realizarlo.

Un paso importante en la inserción de este proyecto, sería utilizar esta aplicación sobre componentes domóticos reales. Esto conllevaría un alto coste en tiempo ya que se necesitaría adaptar todos los datos enviados y/o recibidos por los diferentes elementos. Posiblemente, se pudiera realizar una única plataforma que realizara la serialización de estos datos, para que fueran entregados a la aplicación. Una vez estuvieran tratados y fuera necesaria una nueva comunicación, tan sólo sería necesario volver a transformar los datos para llegar a las diferentes entidades.

5.4. Valoración Freepastry

Freepastry presenta unas opciones, muy valoradas hoy en día pues, ante todo, se presenta al mercado como un modelo *open-source*, imposibilitando su comercialización.

Además, cabe destacar que cuenta con un equipo de desarrollo con una alta actividad. Este equipo permite además, la suscripción a una lista de correo donde asesoran a los programadores con problemas en esta tecnología.

Paralelamente al grupo de Freepastry, existen diversos proyectos paralelos que trabajan sobre esta tecnología al igual que Past. Entre estos proyectos se encuentran algunos dirigidos desde empresas y universidades reconocidas a

nivel mundial. Seguidamente, se muestra un listado de las diferentes tecnologías creadas sobre freepastry, exponiendo a su vez, una breve explicación de las facilidades que otorgan:

- **Scribe:** Sistema de comunicaciones de grupo y notificación de eventos.
- **Squirrel:** Servicio de cache para web, basado en la idea de permitir un buscador en las máquinas para compartir su cache local.
- **SplitStream:** Sistema de alta velocidad para la distribución de streaming o flujos de alto ancho de banda.
- **Post:** Servicio de mensajería. Habitualmente orientado a realizar soporte a ePost i imPost.
- **Scrivener:** Arquitectura para garantizar una justa distribución, a la hora de compartir datos entre nodo.
- **Pasta:** Proyecto del Laboratorio de Computadores de la Universidad de Cambridge, para ofrecer un sistema de ficheros similar a Past pero proporcionando una gestión del espacio de almacenamiento de mejores características.
- **Herald Project:** Desarrollado por el equipo de investigación de Microsoft. Es una plataforma para ofrecer sistemas de suscripción para la notificación de eventos, para trabajar en redes de gran tamaño.
- **Pastiche:** Desarrollado por Ann Arbour en la Universidad de Michigan, propone un sistema de backup vía peer-to-peer.
- **DPSR Project:** En desarrollo por West Lafayette de la Purdue University, esta tecnología explota la sinergia entre los conocimientos de topología estructura peer-to-peer y los protocolos de enrutado multi-hop para redes móviles Ad Hoc.

5.5. Impacto medioambiental

El contenido de este proyecto trata del estudio de redes P2P. El empleo de estas tecnologías en un ámbito doméstico conlleva un control tanto automático como remoto en el suministro de luz. Si esto se extiende a suministros de agua, de gas y ventilación, se consigue la mejora en el aprovechamiento de algunos recursos medioambientales.

Hechos como dejar alguna luz encendida o la calefacción conectada pueden ser monitorizadas y con pequeñas mejoras incluso podrían ser controladas.

5.6. Conclusiones personales

La realización del este proyecto ha sido muy positiva. De esta experiencia me llevo un crecimiento personal y conceptual de gran aprecio para mi persona.

El hecho de trabajar en un proyecto de estas dimensiones con Java, lenguaje que aun conociendo elementalmente no había aplicado en tales magnitudes, me ha llevado a conocer aspectos en su manejo como para asumir cualquier nueva propuesta, conocer nuevas librerías y aprender sus usos de manera más profunda.

Del mismo nodo, haber trabajado con Freepastry y Past, dos proyectos en contínuo desarrollo, me ha permitido entablar comunicación con los desarrolladores y seguir todas las consultas que diferente gente del mundo llevaba a cabo.

Aspectos como el uso de patrones de diseño y los diagramas UML, de los cuales nunca había hecho uso, son algunos de los valores añadidos que también considero positivos para mi futuro.

Incluso todos los problemas que han aparecido en la realización del proyecto, son hoy aspectos positivos pues la resolución de ellos conlleva una experiencia importante.

Pero como aspecto más importante, me llevo todo un proceso de realización de un proyecto. Cada uno de los puntos desde la documentación, el diseño, el desarrollo y la resolución de problemas son la experiencia más importante que obtengo para mi crecimiento.

CAPÍTULO 6. BIBLIOGRAFIA

[1] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Alemania, páginas 329-350, Noviembre de 2001.

[2] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "Scalable Application-level Anycast for Highly Dynamic Groups", *NGC* 2003, Munich, Alemania, Septiembre de 2003.

[4] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", *HotOS VIII, Schoss Elmau*, Alemania , Mayo de 2001.

[5] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", *ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, Octubre de 2001.

[8] Jeff Hoye. "FreePastry Tutorial" [en línea][Última consulta 15 de Febrero de 2008] URL <http://freepastry.org/FreePastry/tutorial/>

[9] FreePastry Team. "FreePastry APIs JavaDoc" [en línea][citado el 20 de Febrero de 2007] URL <http://freepastry.org/FreePastry/javadoc/index.html>

[10] Sun Mycrosistems. "Java 1.5 APIs JavaDoc" [en línea] [Última consulta el 11 de Enero de 2007] URL <http://java.sun.com/j2se/1.5.0/docs/api/>

[11]Web Services-Axis: Web del proyecto de Apache para desarrollar servicios web. Manual de instalación, guía de desarrollo, etc (en línea). [Última Consulta: 27 de Noviembre de 2007]. URL: <http://ws.apache.org/axis/>